(12) NACH DEM VERTRAG ÜBER DIE INTERNATIONALE ZUSAMMENARBEIT AUF DEM GEBIET DES PATENTWESENS (PCT) VERÖFFENTLICHTE INTERNATIONALE ANMELDUNG

(19) Weltorganisation für geistiges Eigentum
Internationales Büro

(43) Internationales Veröffentlichungsdatum
24. Juli 2003 (24.07.2003)

PCT

(10) Internationale Veröffentlichungsnummer
WO 03/060747 A2

(51) Internationale Patentklassifikation[7]: G06F 15/78

(21) Internationales Aktenzeichen: PCT/DE03/00152

(22) Internationales Anmeldedatum:
20. Januar 2003 (20.01.2003)

(25) Einreichungssprache: Deutsch

(26) Veröffentlichungssprache: Deutsch

(30) Angaben zur Priorität:
102 02 044.2 19. Januar 2002 (19.01.2002) DE
102 02 175.9 20. Januar 2002 (20.01.2002) DE

(71) Anmelder *(für alle Bestimmungsstaaten mit Ausnahme von US)*: **PACT XPP TECHNOLOGIES AG** [DE/DE]; Muthmannstrasse 1, 80939 München (DE).

(72) Erfinder; und
(75) Erfinder/Anmelder *(nur für US)*: **VORBACH, Martin** [DE/DE]; Gotthardstrasse 117a, 80689 München (DE). **BAUMGARTE, Volker** [DE/DE]; Barbarossastrasse 14, 81677 München (DE).

(74) Anwalt: **PIETRUK, Claus, Peter**; Patentanwalt, Heinrich-Lilienfein-Weg 5, 76229 Karlsruhe (DE).

(81) Bestimmungsstaaten *(national)*: AE, AG, AL, AM, AT (Gebrauchsmuster), AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ (Gebrauchsmuster), CZ, DE (Gebrauchsmuster), DE, DK (Gebrauchsmuster), DK, DM, DZ, EC, EE (Gebrauchsmuster), EE, ES, FI (Gebrauchsmuster), FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK (Gebrauchsmuster), SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Bestimmungsstaaten *(regional)*: ARIPO-Patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), eurasisches Patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), europäisches Patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, SE, SI, SK, TR), OAPI-Patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Veröffentlicht:
— *ohne internationalen Recherchenbericht und erneut zu veröffentlichen nach Erhalt des Berichts*

Zur Erklärung der Zweibuchstaben-Codes und der anderen Abkürzungen wird auf die Erklärungen ("Guidance Notes on Codes and Abbreviations") am Anfang jeder regulären Ausgabe der PCT-Gazette verwiesen.

WO 03/060747 A2

(54) Title: RECONFIGURABLE PROCESSOR

(54) Bezeichnung: RECONFIGURIERBARER PROZESSOR

(57) Abstract: The invention relates to a processor with a reconfigurable field of data-processing cells and a register means. Said register means is provided with a data stream storage means designed to store a data stream or parts thereof.

(57) Zusammenfassung: Die Erfindung betrifft einen Prozessor mit einem rekonfigurierbaren Feld von datenverarbeitenden Zellen und einem Registermittel. Hierbei ist vorgesehen, daß das Regiestermittel ein Datenstromspeichermittel aufweist, das dazu ausgelegt ist, einen Datenstrom bzw. Teile davon zu speichern.

Rekonfigurierbarer Prozessor

Beschreibung

Die vorliegende Erfindung betrifft das oberbegrifflich Beanspruchte und befaßt sich somit mit rekonfigurierbaren multidimensionalen Logikfeldern und deren Betrieb.

Rekonfigurierbare Elemente werden abhängig von der auszuführenden Applikation unterschiedlich und applikationsentsprechend ausgestaltet. Unter einer rekonfigurierbaren Architektur werden vorliegend Bausteine (VPU) mit konfigurierbarer Funktion und/oder Vernetzung verstanden, insbesondere integrierte Bausteine mit einer Mehrzahl von ein- oder mehrdimensional angeordneten arithmetischen und/oder logischen und/oder analogen und/oder speichernden und/oder intern/extern vernetzenden Baugruppen, die direkt oder durch ein Bussystem miteinander verbunden sind.

Zur Gattung dieser Bausteine zählen insbesondere systolische Arrays, neuronale Netze, Mehrprozessor Systeme, Prozessoren mit mehreren Rechenwerken und/oder logischen Zellen und/oder kommunikativen/peripheren Zellen (IO), Vernetzungs- und Netzwerkbausteine wie z.B. Crossbar-Schalter, ebenso wie bekannte Bausteine der Gattung FPGA, DPGA, Chameleon, VPUTER, etc.. Hingewiesen wird insbesondere in diesem Zusammenhang auf die folgenden Schutzrechte und Anmeldungen desselben Anmelders: DE 44 16 881 A1, DE 197 81 412 A1, DE 197 81 483 A1, DE 196 54 846 A1, DE 196 54 593 A1, DE 197 04 044.6 A1, DE 198 80 129 A1, DE 198 61 088 A1, DE 199 80 312 A1, PCT/DE 00/01869, DE 100 36 627 A1, DE 100 28 397 A1, DE 101 10 530 A1, DE 101 11 014 A1, PCT/EP 00/10516, EP 01 102 674 A1, DE 198 80 128 A1, DE 101 39 170 A1, DE 198 09 640 A1, DE 199 26 538.0 A1, DE 100 50 442 A1, sowie PCT/EP 02/02398, DE 102 40 000, DE 102 02 044, DE 102 02 175, DE 101 29 237, DE 101 42 904, DE 101 35 210, EP 01 129 923, PCT/EP 02/10084, DE 102 12 622, DE 102 36 271, DE 102 12 621, EP 02 009 868, DE 102 36 272, DE 102 41 812, DE 102 36 269, DE 102 43 322, EP 02 022 692, PACT40 (Anmelde-NR. fehlt noch!) . Auf diese Dokumente wird unten durch die anmelderinternen Bezugszeichen Bezug genommen. Diese sind hiermit zu Offenbarungszwecken vollumfänglich eingegliedert.

Die o.g. Architektur wird beispielhaft zur Verdeutlichung herangezogen und im folgenden VPU genannt. Die Architektur besteht aus beliebigen arithmetischen, logischen (auch Speicher) und/oder Speicherzellen und/oder Vernetzungszellen und/oder kommunikativen/peripheren (IO) Zellen (PAEs), die zu einer ein- oder mehrdimensionalen Matrix (PA) angeordnet sein können, wobei die Matrix unterschiedliche, beliebig

ausgestaltete Zellen aufweisen kann; auch die Bussysteme werden dabei als Zellen verstanden. Der Matrix als ganzes oder Teilen davon zugeordnet ist eine Konfigurationseinheit (CT, Ladelogik), die die Vernetzung und Funktion des PA konfiguriert. Die CT kann z. B. als dedizierte Einheit gem. PACT05, PACT10, PACT17, ausgestaltet sein oder als Host-Mikroprozessor nach P 44 16 881.0-53 , DE 102 06 856.9 dem PA zugeordnet bzw. mit oder durch solche realisiert sein.

Die Erfindung beschreibt ein Prozessormodell für rekonfigurierbare Architekturen, das in wesentlichen Punkten an das Modell eines klassischen Prozessors angelehnt ist. Zum besseren Verständnis wird zunächst das klassische Modell näher betrachtet. Dabei wird auf die Betrachtung prozessorexterner Ressourcen (z. B. Hauptspeicher für Programm und Daten etc.) verzichtet.

Ein Prozessor führt in einem *Prozeß* ein *Programm* aus. Das Programm besteht dabei aus einer endlichen Menge von Befehlen (Diese Menge darf Elemente mehrfach enthalten) sowie Informationen über die Reihenfolge, in der Befehle aufeinanderfolgen können. Diese Reihenfolge wird primärüber die lineare Anordnung der Befehle im Programmspeicher und die Ziele von Sprungbefehlen festgelegt.

Befehle werden dabei üblicherweise über ihre Adresse identifiziert. Als Beispiel zeigt Figur 1 (a) ein Programm in VAX-Assembler zur Exponentiation.

Man kann ein Programm auch als gerichteten Graphen auffassen, wobei die Befehle die Knoten bilden, und die Reihenfolge als Kanten des Graphen modelliert wird. Dieser Graph wird in Fi-

gur 1 (b) gezeigt. Der Graph besitzt dabei einen eindeutigen Start- und einen eindeutigen Endknoten. (Im Bild nicht gezeigt, durch die Pfeile angedeutet.) Die Kanten können zusätzlich mit Übergangswahrscheinlichkeiten markiert werden. Diese Information kann dann zur Sprungvorhersage genutzt werden. Die Sprungvorhersage kann wiederum verwendet werden, um Konfigurationen in den Speicher der CT einer VPU vorzuladen (vgl. Patentanmeldung PACT10, die zu Offenbarungszwecken vollumfänglich eingegliedert ist) und/oder Konfigurationen in den Konfigurationsstack einer PAE vorzuladen (gemäß Patentanmeldungen PACT13, PACT17, PACT31, die zu Offenbarungszwecken vollumfänglich eingegliedert sind). Durch das Vorladen von Konfigurationen in den lokalen Speicher der CT (vgl. PACT10, 17) und/oder in den PAE-lokalen Konfigurationscache (PACT17, 31) können die Konfigurationen dann bei Bedarf schneller abgerufen werden, was zu einer erheblichen Effizienzsteigerung führt.

Durch die lineare Anordnung der Befehle im Speicher ergeben sich mehr Abhängigkeiten als unbedingt notwendig. So sind z. B. im gezeigten Beispiel die Befehle DECL und MULL2 voneinander unabhängig. Dies geht aus dem Graphen in Figur 1 (b) nicht hervor. Das Modell kann entsprechend erweitert werden durch Aufteil- und Zusammenfassungsknoten. Dies ist in Figur 1(c) gezeigt.

Heutige Prozessoren erkennen derartige Möglichkeiten der Parallelausführung schon zum Teil in Hardware und verteilen die Operationen auf verschiedene Rechenwerke. Für die weiteren Betrachtungen wird das Modell aus Figur 1(b) verwendet. Die Behandlung der zusätzlichen Komplexität des Aufteilens und Zusammenfassens wird auf einen späteren Zeitpunkt verschoben.

Ein Prozeß braucht zu seiner Ausführung außer dem Programm weitere Ressourcen. Diese sind innerhalb des Prozessors die Register und die Status-Flags.

Diese Ressourcen dienen dazu, Informationen zwischen den einzelnen Befehlen des Programms zu übermitteln. Es ist Aufgabe des Betriebssystems, dafür zu sorgen, daß einem Prozeß die zu seiner Ausführung benötigten Ressourcen zur Verfügung stehen und bei seiner Beendigung wieder freigegeben werden. Heutige Prozessoren besitzen üblicherweise nur einen Registersatz, so daß nur ein Prozeß gleichzeitig auf dem Prozessor ablaufen kann. Es ist nachvollziehbar, daß die Befehle von zwei unterschiedlichen Prozessen in beliebiger Reihenfolge durchmischt ausgeführt werden können, solange beide Prozesse disjunkte Ressourcen verwenden (so, wenn z. B. Prozeß 1 die Register 0-3 und Prozeß 2 Register 4-7 verwendet).
Befehle eines Prozessors haben üblicherweise die folgenden Eigenschaften:
- Ein Befehl wird während der Ausführung nicht unterbrochen.
- Die Ausführungszeit aller Befehle überschreitet einen gewissen Maximalwert nicht.
- Ungültige Befehle werden vom Prozessor erkannt.

Die Aufgabe der Erfindung besteht darin, Neues für die gewerbliche Anwendung bereitzustellen.

Die Lösung der Aufgabe wird unabhängig beansprucht. Bevorzugte Ausführungsformen finden sich in den Unteransprüchen.

2 Übertragung des Modells auf die VPU-Architektur

Die beispielhafte VPU-Architektur ist eine rekonfigurierbare Prozessorarchitektur, die wesentlich durch die Patent(e) (bzw.

Anmeldungen PACT01, 02, 03, 04, 05, 07, 08, 09, 10, 13, 17, 22, 23, 24, 31 definiert ist. Diese Schriften werden, wie vorerwähnt, zu Offenbarungszwecken vollumfänglich eingegliedert. Ebenfalls wird auf PACT11, 20, 27 verwiesen, in denen entsprechende Hochsprachen-Compiler beschrieben sind, sowie auf PACT 21, worin ein entsprechender Debugger beschrieben ist. Auch diese Schriften werden zu Offenbarungszwecken vollständig eingegliedert.

Der klassische Befehl wird ersetzt durch eine Konfiguration im bekannten Sinne, im folgenden Komplex-Befehl (Complex Intruction Word, CIW) genannt. Die Kanten der Graphen in Figur 1 (b) werden realisiert durch Triggersignale an die CT. Damit kann ein vollständiges Programm realisiert werden, indem nach der erfolgten Abarbeitung eines CIW die CT und/oder der Konfigurationscache der PAEs (s. PACT31 und/oder wie nachfolgend beschrieben) das nachfolgende CIW lädt.
Es wurde zunächst erkannt, wie eine Entsprechung von Registern herkömmlicher Prozessoren auf der VPU-Architektur gestaltet werden kann. Dabei wurde herausgefunden, daß eine wesentliche Vorraussetzung zur Registerimplementierung auf folgendem beruht:
- Da die VPU im wesentlichen auf Datenströmen arbeitet, muß ein Register in der Lage sein, einen Datenstrom bzw. Teile davon zu speichern.
- Ein Register muß alloziert und freigegeben werden können. Dabei muß es so lange belegt bleiben wie das Programm auf der VPU läuft. (HW-Unterstützung der Ressourcenverwaltung des Betriebssystems.)
- Gleichzeitiges Lesen und Schreiben (Read-Modify-Write) desselben Registers sollte möglich sein.

Es wird eine angegeben, wie dies in einem Prozessor erreicht werden kann und es wird weiter erfindungsgemäß vorgeschlagen, entsprechend modifizierte RAM-PAEs zu verwenden. Diese sollen zunächst als Register verwendet werden.

Eine ausführliche Beschreibung der Register-PAEs bevorzugt durch erweiterte und/oder modifizierte RAM-PAREs ist nachfolgend in Abschnitt 4 gegeben. Eine Konfiguration (CIW) wird in dem Moment vom Array entfernt, in dem sie über einen Trigger an die CT das nächste CIW anfordert. Der Reconfig-Trigger (vgl. PACT08) kann dabei entweder über den Reconfig-Port einer ALU-PAE oder implizit durch die CT erzeugt werden. In optimal ausgestalteten Versionen sollte dies grundsätzlich von der CT aus erfolgen.

Ebenso wie Befehle auf einem klassischen Prozessor nicht unterbrochen werden, läuft bevorzugt auch ein CIW auf der VPU ohne Unterbrechung bis es das nächste CIW über einen Trigger an die CT anfordert. Es wird nicht vorzeitig beendet. Um dennoch einen regelmäßigen Befehlswechsel sicherstellen zu können (dieser wird später für Multitasking benötigt), wird die maximale Ausführungszeit eines CIW nach oben beschränkt. Damit wird die zweite Eigenschaft eines Befehls gefordert. Es ist bevorzugt die Aufgabe des Compilers, dafür zu sorgen, daß jedes erzeugte CIW dieser Bedingung genügt. Ein CIW, das diese Bedingung verletzt, ist ein ungültiger Befehl. Er kann von der Hardware während der Ausführung z. B. über einen Watchdog-Timer der nach Ablauf einer bestimmten Zeit einen Trigger quasi als Warnsignal generiert, erkannt werden.

Bevorzugt wird das Warnsignal als TRAP durch die Hardware und/oder das Betriebssystem verwaltet. Ebenfalls bevorzugt wird das Signal an die CT gesendet. Ein ungültige CIW bevor-

zugt über einen Reconfig-Trigger, der das reset-ähnliche Löschen sämtlicher Konfigurationen im PA bewirkt, beendet und/oder ebenfalls bevorzugt eine Exception an das Betriebssystem geschickt.

Da die CIWs sehr lang sind, sind dementsprechend auch die Instruction-Fetch-(Zeit zwischen Rekonfigurations-Trigger der PAEs an die CT (vgl. PACT08) und Konfguration ist im FILMO-Cache geladen) und Instruction-Decode-Zeiten (Verteilung der Konfigurations-Daten vom FILMO-Cache (siehe PACT10) in die Konfigurations-Register der PAEs) sehr lang. Dadurch ist die Auslastung der Execution Units (also dem PA im VPU-Prozessormodell) durch einen Prozeß nicht sehr hoch. Wie dieses Problem mit mehreren Prozessen gelöst werden kann, wird nachfolgend in Abschnitt 6 gezeigt.

3 Unterprogramme

Ein Unterprogramm in der Graphendarstellung ist ein Teilgraph eines Programms mit eindeutig bestimmten Eingangsknoten. Die Kante des Unterprogrammaufrufs innerhalb des Graphen ist dadurch statisch bekannt. Die weiterführende Kante am Ausgangsknoten des Unterprogramms ist jedoch nicht statisch bekannt. In Figur 2 wird dies verdeutlicht. Die Kanten vom Hauptprogramm (0201/0202) zum Unterprogramm (0205) sind vorhanden, die Fortführung (0206) nach dem Unterprogramm ist jedoch dem Unterprogramm 0205 nicht bekannt. Die jeweilige Fortführung ist fest mit dem Unterprogrammaufruf verbunden (durch gestrichelte bzw. gepunktete Linien markiert). Sie muß vor dem Erreichen des Eingangsknotens in geeigneter Weise in den Graphen eingefügt werden (0207, 0208). Dies ist in Figur 3 verdeutlicht.

In klassischen Prozessoren geschieht dies üblicherweise dadurch, daß beim Unterprogrammaufruf (Call, 0203, 0204) die Adresse des auf das Unterprogramm folgenden Befehls (das ist genau die fehlende Kante) auf einem Call-Stack abgelegt wird. Von dort kann sie beim Rücksprung (Return) geholt werden.

Übertragen auf die VPU wird also prinzipiell eine Stack-PAE benötigt. Dies ist genauso wie die Register-PAEs eine Prozeß-Ressource und wird genauso verwaltet. Das CIW, das bei seiner Beendigung den Unterprogrammaufruf veranlaßt, konfiguriert die Rücksprung-Kante auf das Stack-PAE. Durch einen Trigger veranlaßt das letzte CIW des Unterprogramms das Stack-PAE, den obersten Eintrag vom Stack zu entfernen und als Rekonfigurationsaufruf an die CT zu schicken.

Bei der Implementierung eines Stacks kann beispielsweise eine der nachfolgende Methoden angewendet werden:

- Eine Lösung innerhalb der CT. Der Stack wird innerhalb der CT in SOftware oder als dedizierte Hardwareeinheit realsiert. Dabei kann eine spezielle Config-ID (z. B. -1) als Rücksprung reserviert sein. Wenn der CT diese ID erhält, ersetzt er sie durch den obersten Eintrag seines lokal verwalteten Stacks.

- Eine Stack-PAE, die beispielsweise als eine modifizierte RAM-PAE nach PACT13 Fig. 21 aufgebaut sein kann.
Stack-Overflow sowie Stack-Underflow sind Exceptions, die bevorzugt an das Betriebssystem weitergegeben werden.

4 Das Register-PAE

Ein klassisches Prozessorregister enthält zu jedem Zeitpunkt ein Datenwort. Ein Befehl kann den Registerinhalt lesen, schreiben oder verändern (Read-Modify-Write).

Ein VPU-Register wird nun die gleichen Eigenschaften aufweisen, allerdings enthält es statt eines einzelnen Wertes erfindugnsgemäß einen Wertevektor oder Teile davon. Es sit möglich und üblicherweise bevorzugt, daß die Organisation eines VPU-Registers als eine Art FIFO erfolgt. In bestimmten Fällen kann aber auch wahlfreier Zugriff erforderlich werden. Im folgenden werden die drei oben angesprochenen Registerzugriffe im einzelnen erläutert. Dabei wird ein wahlfreier Zugriff hier nicht betrachtet.

Lesezugriff Beim Start eines CIW enthält das Register einen Datenvektor unbekannter Länge. Die einzelnen Elemente des Vektors werden sequentiell entnommen. Dabei wird beim letzten Element des Vektors ein Trigger generiert, der anzeigt, dass das Register jetzt leer ist und das CIW terminieren kann. Der Zustand des Registers kann dabei mit 3 Zeigern charakterisiert werden, sie zeigen auf den ersten (0403), letzten (0401) und aktuellen (0402) Eintrag im Datenvektor. Die Stellung der Zeiger zu Beginn eines CIW wird beispielhaft in Figur 4 (a) gezeigt. Dabei steht der Zeiger für den aktuellen Eintrag auf dem ersten Eintrag.

Figur 4 (b) zeigt in einem Beispiel, wie die Zeigerstellung eines Registers am Ende eines CIW aussehen kann. Im dort gezeigten Fall wurde der Vektor nicht vollständig gelesen.

Folglich muß entschieden werden, was mit dem Registerinhalt geschieht. Es gibt bevorzugt die folgenden Möglichkeiten:
- Das Register wird geleert. Alle nicht verarbeiteten Daten werden gelöscht. Der Zeiger für den aktuellen Eintrag wird auf den letzten Eintrag gesetzt.

- Das Register wird auf den Ursprungszustand zurückgesetzt. Dadurch kann das nächste CIW wieder auf den vollen Datenvektor zugreifen. Der Zeiger für den aktuellen Eintrag wird auf den ersten Eintrag zurückgesetzt.
- Nur die bereits gelesenen Daten werden aus dem Register entfernt. Die ungelesenen Daten stehen für das nächste CIW zur Verfügung. Die Zeiger werden dabei nicht verändert.
Im Anschluß daran werden die Werte zwischen dem ersten Eintrag und dem aktuellen Eintrag aus dem Register entfernt. Sie stehen für weitere Operationen nicht mehr zur Verfügung.

Die dritte Möglichkeit ist insbesondere dann interessant, wenn ein CIW aufgrund der maximalen Ausführungszeit für ein CIW den Datenvektor nicht vollständig verarbeiten kann. Siehe hierzu auch Abschnitt 7.

Schreibzugriff Hier werden Daten sequentiell in das Register geschrieben. Dabei wird ein Trigger generiert, wenn der Füllstand des Registers einen bestimmten Wert erreicht. Je nach CIW kann dies eine der folgenden bevorzugten Möglichkeiten sein:
- Das Register ist vollständig gefüllt.
- Es sind noch genau n Einträge im Vektor frei. Dies berücksichtigt die Latenzzeit im CIW, durch die noch n Werte nach dem Trigger auf das Register laufen.
- Das Register ist zu m% gefüllt.

Ein CIW, das versucht, auf ein vollständig gefülltes Register zu schreiben, ist ungültig und wird mit einer Exception beendet (Illegal Opcode). Beim Start des CIW soll festgelegt sein, in welchem Zustand sich das Register befindet. Figur 5 (a) zeigt ein Register vor einem Schreibzugriff, das noch Daten enthält. Es wird vorgeschlagen, daß bestehende Daten kön-

nen gelöscht werden, so daß der Schreibzugriff mit einem leeren Vektor beginnt (Figur 5 (b)). Alternativ können die geschriebenen Daten auch an den bestehenden Inhalt angehängt werden. Dies zeigt Figur 5 (c). Dies ist dann interessant, wenn das vorhergehende CIW nicht den kompletten Vektor erzeugen konnte aufgrund der maximalen Ausführungszeit.

Der Zustand des Registers nach erfolgter Schreiboperation zeigt Figur 5 (d) bzw. (e). Die neu geschriebenen Daten sind dabei schraffiert gekennzeichnet.

paralleler Schreib-/Lesezugriff Die Beschränkung auf reine Schreib- bzw. Lesezugriffe erfordert eine höhere Registerzahl als nötig. Wenn einem Register durch Lesezugriffe Daten entnommen werden, entsteht dadurch Platz, in dem Schreibdaten aufgenommen werden können. Es muß lediglich sichergestellt werden, dass geschriebene Daten nicht vom gleichen CIW wieder gelesen werden können, d.h. dass eine klare Trennung zwischen den Lesedaten eines CIWs und den Schreibdaten des CIWs existiert. Dazu wird in den FIFO eine virtuelle Trennlinie (0601) eingeführt. Das Register wurde vollständig gelesen, wenn diese Trennlinie am Ausgang des FIFOs angelangt ist. Zur Festlegung dieser virtuellen Trennlinie können geeignete Mittel implementiert sein.

Kann ein Schreibzugriff für ein Datenwort nicht ausgeführt werden, weil das Register noch mit ungelesenen Lesedaten blockiert ist, wird das CIW beendet und eine Illegal Opcode Exception erzeugt. Das Verhalten des Registers ist ansonsten genauso wie bei Schreib- und Lesezugriff erläutert.
Zusätzlich soll spezifiziert werden, was mit der virtuellen Trennlinie zwischen Lese- und Schreibdaten geschieht. Diese kann entweder an der Stelle verbleiben, wo sie gerade steht.

Dies ist dann nützlich, wenn ein CIW aufgrund der Zeitbeschränkung beendet werde muß. Alternativ wird die Trennlinie an das Ende aller Daten gesetzt.

Kombinierte Schreib-/Lesezugriffe sind allerdings problematisch, wenn das CIW mit einer Exception beendet wurde. In diesem Fall ist es nicht mehr ohne weiteres möglich, die Register auf ihre Werte beim Start des CIW zurückzustellen. Das Debugging kann dadurch zumindest erschwert werden, siehe auch nachfolgend in Abschnitt 8.

Figur 6 zeigt die Funktionsweise an einem Beispiel. 0601 kennzeichnet die virtuelle Trennlinie. Zu Beginn enthält das Register Daten (a), die im folgenden teilweise (b) bzw. vollständig (c) gelesen werden. Neu geschriebene und gelesene Einträge sind dabei durch unterschiedliche Schraffur gekennzeichnet. Die Teilbilder (d) und (e) zeigen den Zustand des Registers nach dem notwendigen Zeiger-Update, das die Trennlinienlage verändert. Dies ist kein expliziter Schritt, sondern wird hier nur zur Verdeutlichung dargestellt. Die gelesenen Einträge müssen sofort entfernt werden, um Platz für die neu zu schreibenden Einträge zu schaffen.

Ein Prozeß, also ein Programm, das auch insbesondere bei einem Multitasking-Betrieb Ressourcen mit anderen Programmen teilt, muß jedes benötigte Register allozieren, bevor er es verwenden kann. Dies geschieht bevorzugt über ein zusätzliches Konfigurationsregister innerhalb der RAM- und/oder Register-PAE. Dort wird auch eingetragen, zu welchem Prozeß das Register jetzt gehört. Diese Konfiguration bleibt auch über Rekonfigurationen hinweg erhalten. Das Register muß explizit vom CT freigegeben werden. Dies geschieht beispielsweise bei der Beendigung eines Prozesses. Mit der Konfiguration jedes

CIWs muß den Registern mitgeteilt werden, zu welchem Prozeß das CIW gehört. Dies ermöglicht das Umschalten zwischen mehreren Registersätzen. Das Verfahren wird nachfolgend genauer im Abschnitt 6 beschrieben.

5 Interrupts

Bei Interrupts muß zwischen zwei unterschiedlichen Typen unterschieden werden. Zum einen gibt es die Hardware-Interrupts, wo der Prozessor auf ein externes Ereignis reagieren muß. Diese werden üblicherweise vom Betriebssystem bearbeitet und sind für die laufenden Prozesse nicht sichtbar. Sie sollen hier nicht weiter behandelt werden. Der zweite Typ sind die Software-Interrupts. Diese werden häufig benutzt, um asynchrone Interaktionen zwischen Prozeß und Betriebssystem zu realisieren. So ist es z. B. unter VMS möglich, eine Leseanforderung an das Betriebssystem zu schicken, ohne auf die eigentlichen Daten zu warten. Sobald die Daten vorhanden sind, unterbricht das Betriebssystem das laufende Programm und ruft asynchron eine Prozedur des Programms auf. Dieses Verfahren nennt sich dort Asynchronous System Trap (AST).

Dieses Verfahren kann in gleicher Form auf der VPU angewendet werden. Hierzu kann Unterstützung im CT vorgesehen werden. Der CT weiß, ob eine asynchrone Routine für einen Prozeß aufgerufen werden muß. In diesem Fall wird der nächste Request, der vom Array kommt, nicht direkt abgearbeitet, sondern gespeichert.

Stattdessen wird eine Folge von CIWs eingefügt, die zunächst die Prozessorstatus (die Registerinhalte) sichern, die asynchrone Routine ausführen und dann die Registerinhalte wiederherstellen. Im Anschluß daran kann der ursprüngliche Request

abgearbeitet werden.

6 Multitasking

In Abschnitt 2 wurde festgestellt, daß die VPU-Architektur mit nur einem Prozeß unter Umständen nicht optimal ausgelastet werden kann, weil etwa aufgrund der Länge der CIWs sehr hohe Lade- und Dekodierungszeiten auftreten. Dieses Problem kann durch die gleichzeitige Ausführung mehrerer Prozesse gelöst werden. Hierbei werden auf der VPU nun erfindungsgemäß mehrere Registersätze vorgesehen, was erlaubt, daß beim Kontextwechsel einfach zwischen den Registersätzen umgeschaltet werden kann und keine aufwendigen Register-Freiräum- und -Lade-Operationen erforderlich werden. Die Verarbeitungsgeschwindigkeit kann so erhöht werden.

Während der Ausführung von CIWs der Prozesse steht genügend Zeit zur Verfügung, um den nächsten Befehl des aktuellen Prozesses zu holen und über den FILMO an die PAEs zu verteilen, bzw. aus dem Konfigurationscache in die PAEs zu laden (vgl PACT31). Die optimale Anzahl an Registersätzen kann dabei in Abhängigkeit von der durchschnittlichen Ausführungszeit eines CIW und den durchschnittlichen Lade- und Dekodierzeiten der CIWs bestimmt werden.

Dabei kann Latenzzeit durch eine größere Anzahl Registersätze abgefangen werden. Wichtig für die Funktion des Verfahrens ist, daß die durchschnittliche CIW-Laufzeit größer ist als die jeweils effektiv benötigte Zeit zum Laden bzw. Dekodieren des CIW.

Die korrespondierenden Register der unterschiedlichen Registersätze liegen dabei für den Programmierer auf derselben

PAE-Adresse. D.h. es können zu jedem Zeitpunkt immer nur die Register eines Registersatzes verwendet werden. Der Kontextwechsel zwischen den Registersätzen kann dadurch realisiert werden, dass vor jedem CIW der entsprechende Kontext an die PAEs übertragen wird. Der Kontextswitch kann im Detail durch die PUSH/POP Operationen nach PACT11 und/oder durch eine besondere RAM-/Register-PAE Hardware wie in PACT13 Fig. 21 dargestellt automatisch erfolgen. In beiden Fällen ergibt sich ein ähnlicher Stack-Aufbau im Speicher. Jeder Stack-Eintrag speichert die Daten eines Prozesses. Ein Stackeintrag umfaßt dabei den kompletten Inhalt aller Registers, mit anderen Worten sämtlicher Speicherzellen aller Speicher die als Register für einen Prozess dienen. Ebenso kann gemäß PACT11 ein Stackeintrag auch PA-interne Daten und Zustände enthalten.

Im allgemeinen werden nun auf einem System mehr Prozesse vorhanden sein als Registersätze auf dem Prozessor. Das bedeutet, daß gelegentlich ein Prozeß vom Prozessor entfernt werden muß. Dazu wird ähnlich wie beim Software-Interrupt eine Kante den Programmgraphen vom CT aufgetrennt. Die Registerinhalte des Prozesses werden gesichert und die vom Prozeß allozierten Prozessorressourcen i(Register, Stack-PAE, etc.) wieder freigegeben. Die so freigewordenen Ressourcen werden nun von einem anderen Prozeß alloziert. Dann werden die für diesen Prozeß gespeicherten Registerinhalte wieder zurückgeschrieben und der Prozeß an dessen aufgetrennter Kante fortgesetzt. Das Sichern und Rückladen der Registerinhalte kann dabei über CIWs erfolgen.

7 CIW und Schleifen

Aufgrund der oben geforderten Eigenschaft, daß ein CIW spätestens nach einer gewissen Maximalanzahl an Takten terminieren muß, können allgemeine Schleifen nicht ohne weiteres in ein CIW übersetzt werden. Es ist immer möglich, den Schleifenrumpf in ein CIW zu übersetzen und die Schleifenkontrolle über Rekonfiguration abzuwickeln. Dies kostet jedoch oftmals erheblich Performance. Dieser Abschnitt zeigt, wie eine Schleife so umgeformt werden kann, daß die Anzahl der Rekonfigurationen minimiert wird.

Im folgenden wird von folgendem Programmstück ausgegangen:

```
while (condition) {
something;
}
```

Dabei soll sowohl die Laufzeit von condition wie something bestimmt oder nach oben abgeschätzt werden können. Die Schleife kann nun wie folgt umformuliert werden:

```
while (1) {
if (!condition) goto finish;
something;
}
finish:
```

Nun kann der Schleifenrumpf so oft iteriert werden, wie es die maximale Laufzeit eines CIW zuläßt. Hierzu wird eine neue Variable z eingeführt, die weder in condition noch in something vorkommt. Das Programm sieht nun folgendermaßen aus:

```
while (1) {
for (z=0; z<MAX; z++) {
```

```
if (!condition) goto finish;
something;
}
}
finish:
```

Die for-Schleife besitzt eine vom Compiler bestimmbare maximale Laufzeit. Sie kann deshalb auf ein CIW abgebildet werden. MAX wird vom Compiler in Abhängigkeit von der maximalen Laufzeit und den Einzellaufzeiten der Anweisungen bestimmt.

Das so entstehende CIW hat zwei Ausgangskanten. Der Ausgang über das goto führt zum nächsten CIW, der Ausgang über das reguläre Ende des for bildet eine Kante auf sich selbst. Darüber wird die Endlosschleife realisiert.

8 Debugging

Im klassischen Prozessor erfolgt das Debugging auf Befehlsbasis, d. h. der Ablauf eines Programms kann jederzeit zwischen zwei Befehlen unterbrochen werden. An diesen Unterbrechungspunkten hat der Programmierer Zugriff auf die Register. Er kann sie ansehen und modifizieren. Unterbrechungspunkte können auf verschiedene Art und Weise realisiert werden. Zum einen kann das Programm modifiziert werden, d. h. der Befehl, vor dem angehalten werden soll, wird durch andere Befehle ersetzt, die den Debugger aufrufen. Im Graphenmodell entspricht dies dem Austausch eines Knotens durch einen anderen Knoten oder einen Teilgraphen. Eine andere Methode beruht auf zusätzlicher Hardware-Unterstützung. Hierbei wird dem Prozessor mitgeteilt, bei welchem Befehl das Programm unterbrochen wer-

den soll. Der entsprechende Befehl wird dabei üblicherweise über seine Adresse identifiziert.

Beide Möglichkeiten sind auf die VPU erfindungsgemäß übertragbar. Ein CIW kann etwa vom Debugger durch ein anderes CIW ersetzt werden. Dieses CIW kann z. B. die Registerinhalte in den Hauptspeicher kopieren, wo diese entweder mit einem VPU-externen Debugger analysiert werden können. Alternativ kann der Debugger auch auf der VPU ablaufen.

Weiterhin kann eine Hardware-Unterstützung im CT vorgesehen werden, die CIWs bei deren Request anhand der ID identifiziert und dann den Debugger aufruft. Zusätzlich kann eine Unterbrechung auch ein einer Kante des Graphen festgemacht werden, da diese im Gegensatz zu klassischem Programmcode explizit vorliegen.

Die oben beschriebene Art des Debugging ist für klassische Prozessoren vollständig ausreichend, da die Befehle zumeist sehr einfach sind. Eine hinreichend feine Auflösung der beobachtbaren Punkte ist gegeben. Weiterhin kann sich der Programmierer darauf verlassen, daß die einzelnen Befehle korrekt sind. (Dafür sorgt üblicherweise der Prozessorhersteller.)

Auf der VPU hingegen ist es dem Programmierer möglich, sich die CIWs zu definieren, welche eine Art "Prozessorbefehle" bilden. Dementsprechend könnten die so definierten in sich Befehle fehlerhaft sein. Ein Debugging der einzelnen Befehle wird also bevorzugt auf eine im folgenden als Microcode-Debugging bezeichnete Weise ausgelegt. Microcode-Debugging ist so ausgestaltet, daß der Programmierer Zugriff auf alle internen Register und Datenpfade des Prozessors erhält; es

wurde erkannt, daß der dafür erforderliche Aufwand ohne weiteres durch die gesteigerte Funktionalität zu rechtfertigen ist.

Eine Hardware-Unterstützung hierfür ist möglich, aber sehr aufwendig und zu reinen Debugging-Zwecken nicht sinnvoll. Alternativ wird daher der Zustand des Prozessors vor dem fraglichen Befehl gesichert werden und die Ausführung des eigentlichen Befehls auf einem Software-Simulator erfolgen. Dies ist die nach PACT11 bevorzugte Methode zum Debuggen von VPUs. Die Daten und Zustände werden über Businterface, Speicher und/oder bevorzugt über Debug-Interface wie z.B. JTAG an den Debugger übertragen. Bevorzugt kommt ein Debugger nach PACT21 zum Einsatz, der zur Abarbeitung des Micro-Debuggings bevorzugt einen Mixed-Mode Debugger mit integriertem Simulator enthält.

Bei geeignetem Programmiermodell kann der Debugger auch bei Auftreten einer Exception innerhalb eines Befehls aufgerufen werden. Hierzu ist es sinnvoll, dass die Register auf den Zustand vor dem Start des Befehls zurückgestellt werden können und sonst keine Seiteneffekte aufgetreten sind. Dann kann der fragliche Befehl im Software-Simulator gestartet werden und bis zum Auftreten der Exception simuliert werden.

Besonders bevorzugte Debugging-Mechanismen sind in PACT21 ausführlich beschrieben.

Microcode-Debugging kann bevorzugt dadurch realisiert werden, dass nach oder während der Abarbeitung eines CIW ein Debug-CIW konfiguriert wird, das zunächst sämtliche Zustände (z.B. in den PAEs) erhält und diese dann durch eine geeignete Konfiguration der Vernetzungsressourcen in einen externen Spei-

cher schreibt. Hierzu können besonders bevorzugt die in PACT11 beschriebenen PUSH/POP Methoden zum Einsatz gelangen. Bevorzugt kann dies über eine Industriestandard-Interface, wie z.B. JTAG erfolgen. Aus dem Speicher oder über das JTAG-Interface kann dann ein Debugger die Daten übernehmen und ggf. in Verbindung mit einem Simulator (vgl. PACT21) auch schrittweise weitersimulieren, wodurch ein Microcode-Debugging ermöglicht wird.

9 Verteilter Konfigurations-Cache

Aufgrund des zentralen Konfigurations-Caches beim FILMO dauert es, wenn ein solches, was nicht zwingend ist, verwendet wird, verhältnismäßig lange, bis eine Konfiguration auf die einzelnen PAEs einer PAC verteilt ist. Dieser Abschnitt beschreibt nun ein bevorzugtes Verfahren, um diese Zeit abzukürzen. Ein ähnliches, alternatives oder zusätzliches Verfahren ist aus PACT31 bereits bekannt und ist zu Offenbarungszwecken vollumfänglich eingegliedert.

Dazu erhält jedes PAE seinen eigenen lokalen Cache. Dieser speichert die Konfigurationsdaten verschiedener Konfigurationen für genau dieses PAE. Auch die Tatsache, daß ein PAE von einer Konfiguration keine Daten erhalten hat, wird gespeichert. Für jede angeforderte Konfiguration kann der Cache dadurch eine der folgenden Aussagen treffen:

- Die Konfigurationsdaten sind im Cache vorhanden.
- Für diese Konfiguration werden keine Daten benötigt.
- Über diese Konfiguration ist nichts bekannt.
- Es werden Konfigurationsdaten benötigt, diese sind jedoch nicht im Cache verfügbar. (z. B. aufgrund der Länge der Konfiguration, RAM-Preload etc.)

Die letzten beiden Aussagen können dabei zusammengefaßt werden. Bei beiden muß der Code oder die Tatsache, daß kein Code benötigt wird, angefordert werden. Ein Auftrag für eine Konfiguration wird dabei vom FILMO als Broadcast auf dem Testbus an alle PAEs verschickt. Wenn alle PAEs die Konfiguration in ihrem lokalen Cache haben, kann diese per Broadcast auf dem Configbus gestartet werden. Im Idealfall erfordert der Start einer Konfiguration also nur die Übertragung eines einzigen Konfigurationswortes.

Sollte ein PAE nicht über die Konfigurationsdaten verfügen, muß dieses an dem FILMO zurückgemeldet werden. Im einfachsten Fall geschieht dies über ein Reject auf der vorhandendenen Leitung. Der FILMO weiß dann aufgrund dieses Signals, daß mindestens einem PAE der PAC die Konfigurationsdaten fehlen. Er kann dann die kompletten Daten übertragen. Alternativ kann auch jedes PAE getrennt einen Request für die Daten auslösen. Hier muß ein geeigneter Kompromiß zwischen der Anzahl der Requests und der Menge der zu übertragenden Konfigurationsdaten gefunden werden. Kleine PAC-Größen sind dabei auch aufgrund der geringeren Latenz auf dem Konfigurationsbus von Vorteil.

Aufbau des Caches

Ein Cache besteht immer aus zwei Teilen. Der eine Teil enthält die eigentlichen Daten (hier die Konfigurationsworte, 0902), der andere Teil enthält Verwaltungsinformation (hier die enthaltenen Konfigurationsnummern sowie ihr Alter, 0901).

Zunächst wird der Verwaltungsteil beschrieben.

Wünschenswert ist, daß die am längsten nicht mehr benutzte Konfiguration aus dem Cache entfernt wird, wenn dies notwen-

dig wird. Solange nur neue Konfigura-tionen angefordert werden, sind die Einträge im FIFO korrekt sortiert. Sollte eine Konfiguration angefordert werden, für die bereits ein Eintrag im FIFO vorhanden ist, muß dieser Eintrag aus dem FIFO entfernt werden. Er wird dann wieder neu am Ende eingefügt. Eine beispielhaft für diesen Zweck modifizierte FIFO-Stufe zeigt Figur 7. Die schraffierten Module sind dabei zusätzlich zu einer normalen FIFO-Stufe nach dem Stand der Technik. Sie vergleichen mittels des Vergleichers (0701) die Konfigurationsnummer des Dateninhaltes der Stufe mit der geforderten Konfigurationsnummer und erzeugen bei Gleichheit ein ack (0702) für die Stufe. Damit werden die Daten der Stufe über den Multiplexer (0703) gelesen und alle anderen Werte rutschen um eine Stufe nach. Die Einträge in diesem FIFO enthalten außer der Konfigurationsnummer weitere Informationen. Dies ist entweder ein Zeiger (Adresse) auf die Konfigurationsdaten oder eine der beiden Möglichkeiten "keine Daten erforderlich" (z. B. als 0 codiert) bzw. "Daten müssen angefordert werden" (z. B. -1). Die Zusammenschaltung mehrerer Stufen zeigt Figur 8. Dabei wird die Read-Chain mit der geforderten Konfigurationnummer und dem Status -1 initialisiert. Dieser Wert kommt unverändert genau dann am Ausgang der Read-Chain heraus, wenn die Konfigurationsnummer nicht im FIFO gespeichert ist. Damit kann der Ausgang der Read-Chain in jedem Fall dazu verwendet werden, die Konfigurationsnummer auf den FIFO zu schreiben. Das Signal ack_in wird dann aktiviert, wenn der FIFO voll ist und sich die gesuchte Konfigurationsnummer nicht im FIFO befindet. Dies ist der einzige Fall, bei welchem der älteste Eintrag aus dem FIFO entfernt werden muß, weil der Verwaltungsspeicher voll ist. Der eigentliche Datenspeicher wird aufgrund der unterschiedlichen Anzahl der Konfigurationswörter pro Konfiguration als verkette Liste organisiert. Andere Im-plementierungen sind denkbar. Eine verket-

tete Liste kann als RAM einfach dadurch implementiert werden, daß zusätzlich zu den Daten jeweils die Adresse des nachfolgenden Datenwortes abgelegt wird.

Zusätzlich zu den Listen für die eigentlichen Konfigurationen wird eine Frei-Liste geführt, in der alle nicht benutzten Einträge aufgeführt sind. Diese muß nach einem Reset zunächst initialisiert werden.

Figur 9 zeigt einen möglichen Cache-Inhalt während des Betriebs. Freie Einträge im Datenspeicher sind dabei weiß, von einer Konfiguration belegte schraffiert unterlegt. Konfigurationen müssen dabei nicht auf aufeinanderfolgenden Adressen liegen. Konfiguration 18 besitzt keine Konfigurationsdaten, deshalb führt auch kein Zeiger in den Datenspeicher.

Eine neue Konfiguration wird auf die Freiliste in den Datenspeicher geschrieben. Dabei wird die Zeigerinformation des Datenspeichers nicht modifiziert. Lediglich für das letzte Datenwort ein Konfiguration wird die Zeigerinformation verändert, um anzuzeigen, daß die Liste jetzt hier ändert. Der Zeiger auf die Freiliste wird auf den nächsten Eintrag gesetzt.

Es kann vorkommen, dass der Platz in der Freiliste nicht ausreicht, um die ankommenden Konfigurationsdaten vollständig aufzunehmen. In diesem Fall muß entschieden werden, ob eine alte Konfiguration aus dem Datenspeicher entfernt wird oder ob die aktuelle Konfiguration nicht in den Cache aufgenommen werden soll. Im letzteren Fall werden die nachfolgenden Konfigurationswörter verworfen. Da keine Zeiger verändert wurden, bleibt die Freiliste wie zuvor, lediglich einige unbenutzte Datenwörter besitzen einen anderen Wert. Die Entschei-

dung, welche Konfiguration nicht mehr im Cache liegen soll (die älteste oder die aktuelle) kann an der Anzahl der bereits geschriebenen Konfigurationsworte festgemacht werden. Es macht wenig Sinn, mehrere gecachete Konfigurationen zu entfernen, um Platz für z. B. eine lange RAM- Initialisierung zu schaffen.

Wenn die älteste Konfiguration entfernt werden soll, wird sie dem FIFO entnommen. Der Zeiger des letzten Eintrags der Freiliste wird auf den dem FIFO entnommenen Wert gesetzt. Ab dieser Adresse kann nun in der gewohnten Weise weiterkonfiguriert werden.

Figur 10 zeigt dies an einem Beispiel. Neu konfiguriert werden soll die Konfiguration mit der Nummer 7. In Figur (a) ist die Freiliste vollständig belegt worden. Es wird entschieden, die älteste Konfiguration (Nr. 5) vom Cache zu entfernen und Konfiguration Nr. 7 weiter auf den Cache zu schreiben. Dazu wird der Zeiger am Ende der Freiliste auf den Start der ehemaligen Konfiguration 5 umgesetzt. Dadurch wird die Freiliste wieder verlängert und es steht wieder Platz für neue Konfigurationsworte zur Verfügung. Die in diesem Schritt betroffenen Speicherteile sind in Figur (b) gegendiagonal schraffiert. Bei geeigneter Aufteilung des Speichers kann dies in einem Takt erfolgen. Mit dem letzten Konfigurationswort wird der entsprechende Zeiger auf Ende gesetzt und der Freizeiger auf den nächsten Eintrag. Platz im Datenspeicher wird nicht nur dann wieder freigegeben, wenn dieser durch die Aufnahme einer neuen Konfiguration benötigt wird. Auch wenn der Verwaltunsspeicher voll ist und deswegen ein Eintrag vom Verwaltungsspeicher entfernt wird, muß die Freiliste im Datenspeicher angepaßt werden. Hierzu muß entweder der Zeiger am Ende der Freiliste oder am Ende der freiwerdenden Konfiguration ange-

paßt werden. Beide Informationen stehen bisher nicht zur Verfügung. Man kann sich jetzt durch eine der Listen durchbewegen, bis das Ende erreicht wird. Dies ist jedoch zeitaufwendig. Alternativ wird im Verwaltungsspeicher ein zusätzlicher Zeiger auf das jeweilige Ende einer Konfiguration abgelegt. Nun ist die Modifikation einfach möglich. Der Freizeiger erhält die Startadresse der alten Konfiguration, und der Zeiger beim letzten Konfigurationswort im Datenspeicher wird auf den Freizeiger gesetzt.

Dies wird in Figur 11 verdeutlicht. Die Zeiger auf die Konfigurationsenden sind dabei gestrichelt eingezeichnet. UnterFigur (a) zeigt die Situation vor dem Löschen, UnterFigur (b) die danach.

10 Optimierung der Busallokation

Derzeit werden die Busse explizit vom Router festgelegt. Das kann dazu führen, daß zwei Konfigurationen sich auf einem Bus überlappen und deswegen nicht gleichzeitig ablaufen können, obwohl insgesamt genügend Busse zur Verfügung stünden.

Es wurde erkannt, daß es algorithmisch unerheblich ist, über welchen Bus eine Verbindung geführt wird. Deshalb wird vorgeschlagen, eine Busallokation dynamisch von der Hardware durchführen zu lassen und die Hardware mit einem geeigneten dynamsichen Busallokator zu versehen. Eine Konfiguration spezifiziert dazu nur noch, daß sie eine Verbindung von Punkt A nach Punkt B innerhalb einer Zeile benötigt. Welcher der vorhandenen Busse dann tatsächlich verwendet wird, wählt ein Arbiter in der Hardware aus. Dieser kann pro Zeile entweder verteilt über Nachbarschaftsbeziehungen oder an zentraler Stelle für die Zeile arbeiten. Zusätzlich können Busse dyna-

misch umgelegt werden. Dabei können zwei kurze nichtüberlappende Busse, die aufgrund früherer Belegung auf unterschiedliche Busnummern konfiguriert wurden, bei Freiwerden von Ressourcen auf dieselbe Busnummer umgelegt werden. Dadurch wird Platz geschaffen für zukünftige längere Verbindungen.

Rekonfigurierbarer General Purpose Prozessor

Patentansprüche

1. Prozessor mit einem rekonfigurierbaren Feld von datenverarbeitenden Zellen und einem Registermittel, dadurch gekennzeichnet, daß das Registermittel ein Datenstromspeichermittel aufweist, das dazu ausgelegt ist, einen Datenstrom bzw. Teile davon zu speichern.

2. Prozessor nach dem vorhergehenden Anspruch, dadurch gekennzeichnet, daß ein Registermittelalloziermittel und/oder ein Registermittelfreigabemittel vorgesehen ist.

3. Prozessor nach einem der vorhergehenden Ansprüche, dadurch gekennzeichnet, daß das Registeralloziermittel dazu ausgebildet ist, über Rekonfigurationen hinweg erhalten zu bleiben.

4. Prozessor nach einem der vorhergehenden Ansprüche, dadurch gekennzeichnet, daß als Registermittel eine, gegebenenfalls modifizierte RAM-PAE, vorgesehen ist.

5. Prozessor nach einem der vorhergehenden Ansprüche, dadurch gekennzeichnet, daß für Schreib- und Lesezugriff ausgebildete Registermittel vorgesehen sind, insbesondere unter Implementierung eines virtuellen FIFO-Trennlinienmittels.

6. Prozessor nach einem der vorhergehenden Ansprüche, dadurch gekennzeichnet, daß weiter zumindest eine Speichereinheit vorgesehen ist, die zur Verwendung als Stack ausgebildet ist, wobei sie insbesondere dazu ausgebildet ist, einen Stack-Under- und/oder Overflowzustand anzuzeigen, insbesondere einer Betriebssystemeinheit.

Fig. 1a

```
MOVL #1,R2
10$: TSTL R1
     BEQL 20$
     MULL2 R2,R0
     DECL R1
     BRB 10$
20$:
```

Fig. 1b

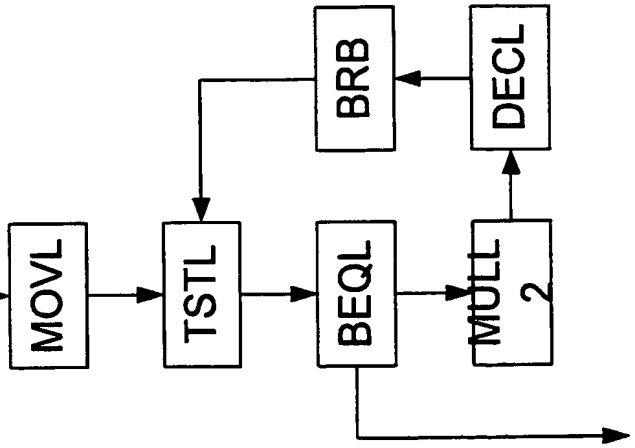
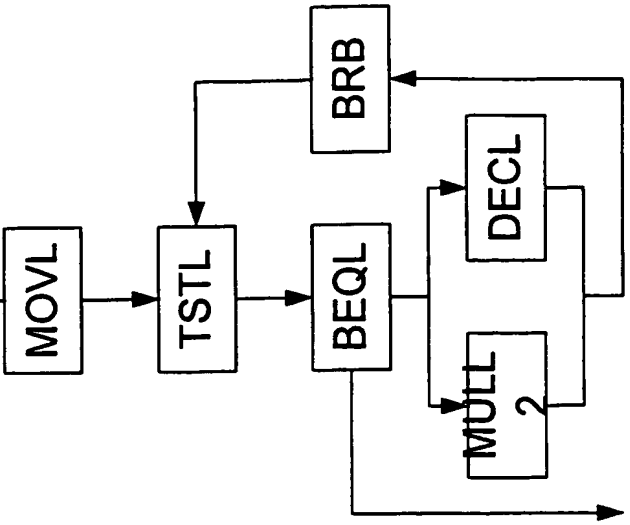


Fig. 1c



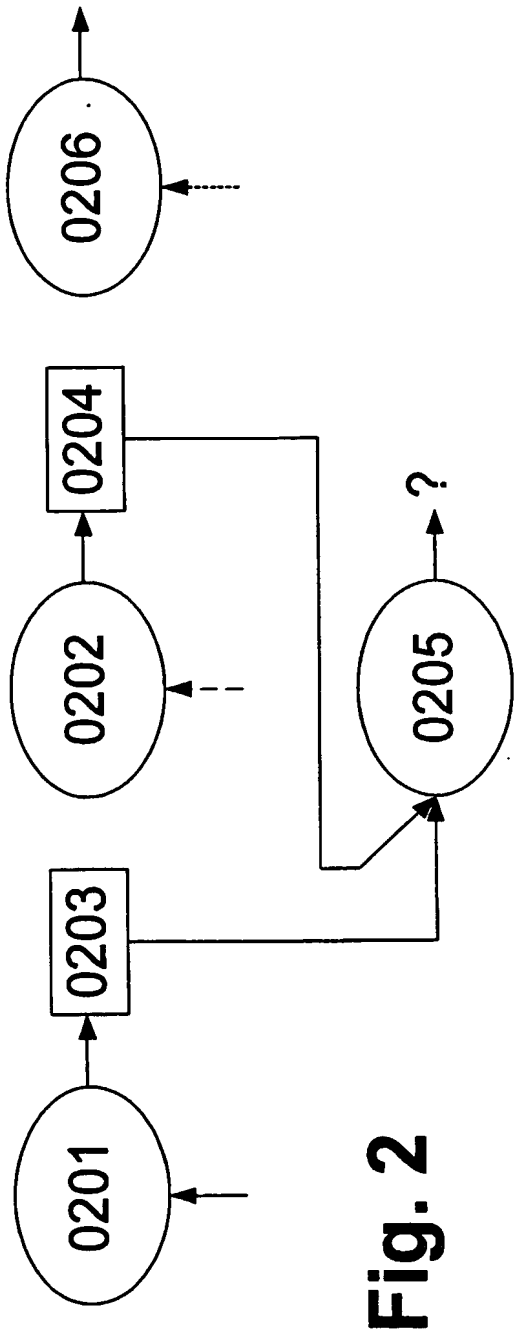

0201
0203
0202
0204
0205
?
0206

Fig. 2

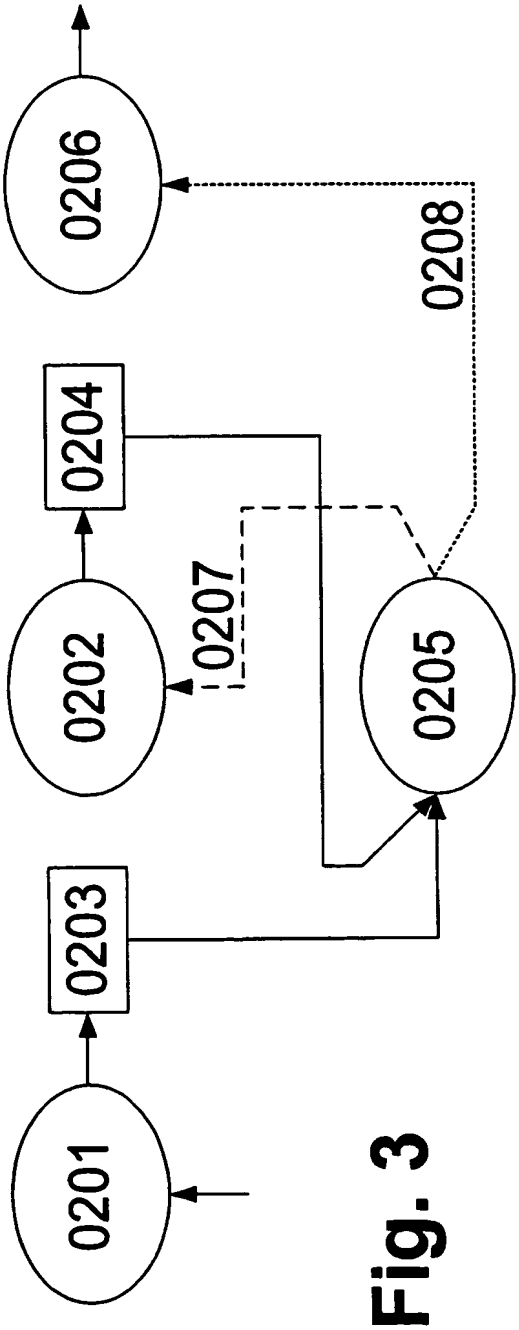

0201
0203
0202
0207
0204
0205
0208
0206

Fig. 3

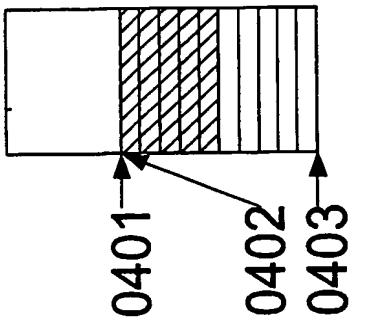


Fig. 5e

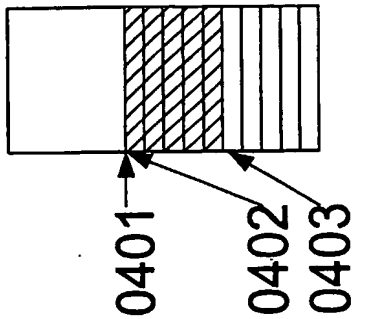


Fig. 5d

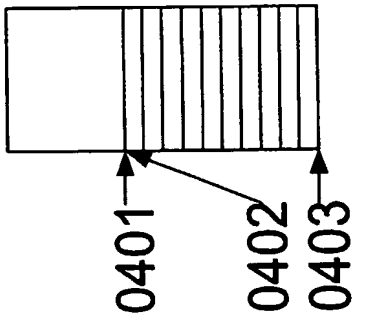


Fig. 5c

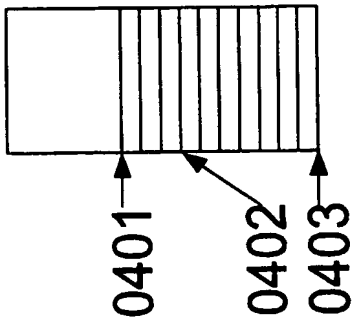


Fig. 4b

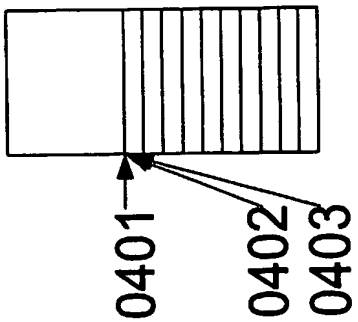


Fig. 5b

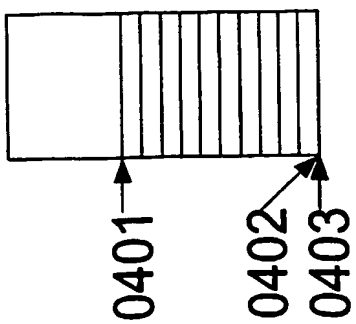


Fig. 4a

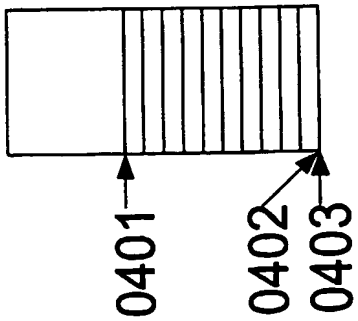


Fig. 5a

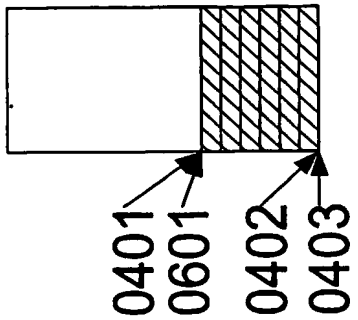


Fig. 6e

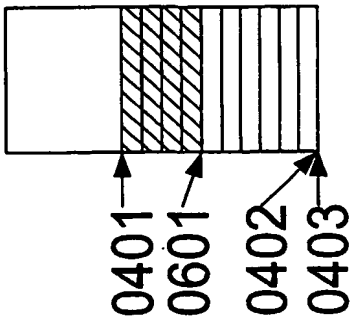


Fig. 6d

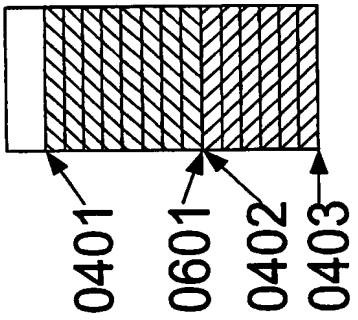


Fig. 6c

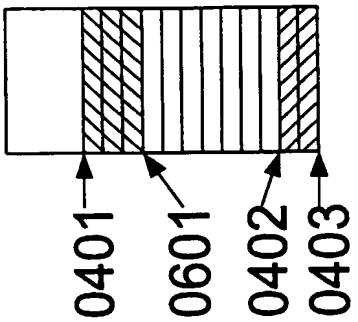


Fig. 6b

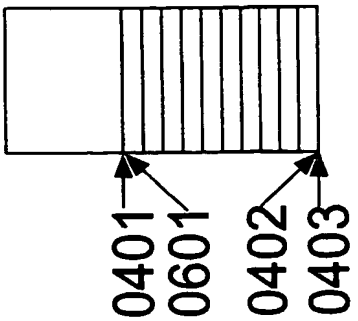


Fig. 6a

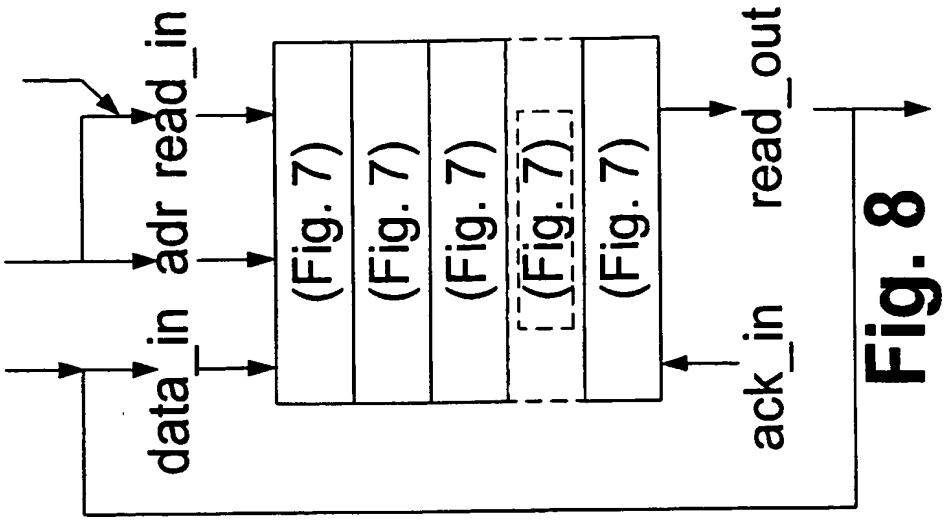

read_in
adr
data_in
(Fig. 7)
(Fig. 7)
(Fig. 7)
(Fig. 7)
(Fig. 7)
read_out
ack_in

Fig. 8

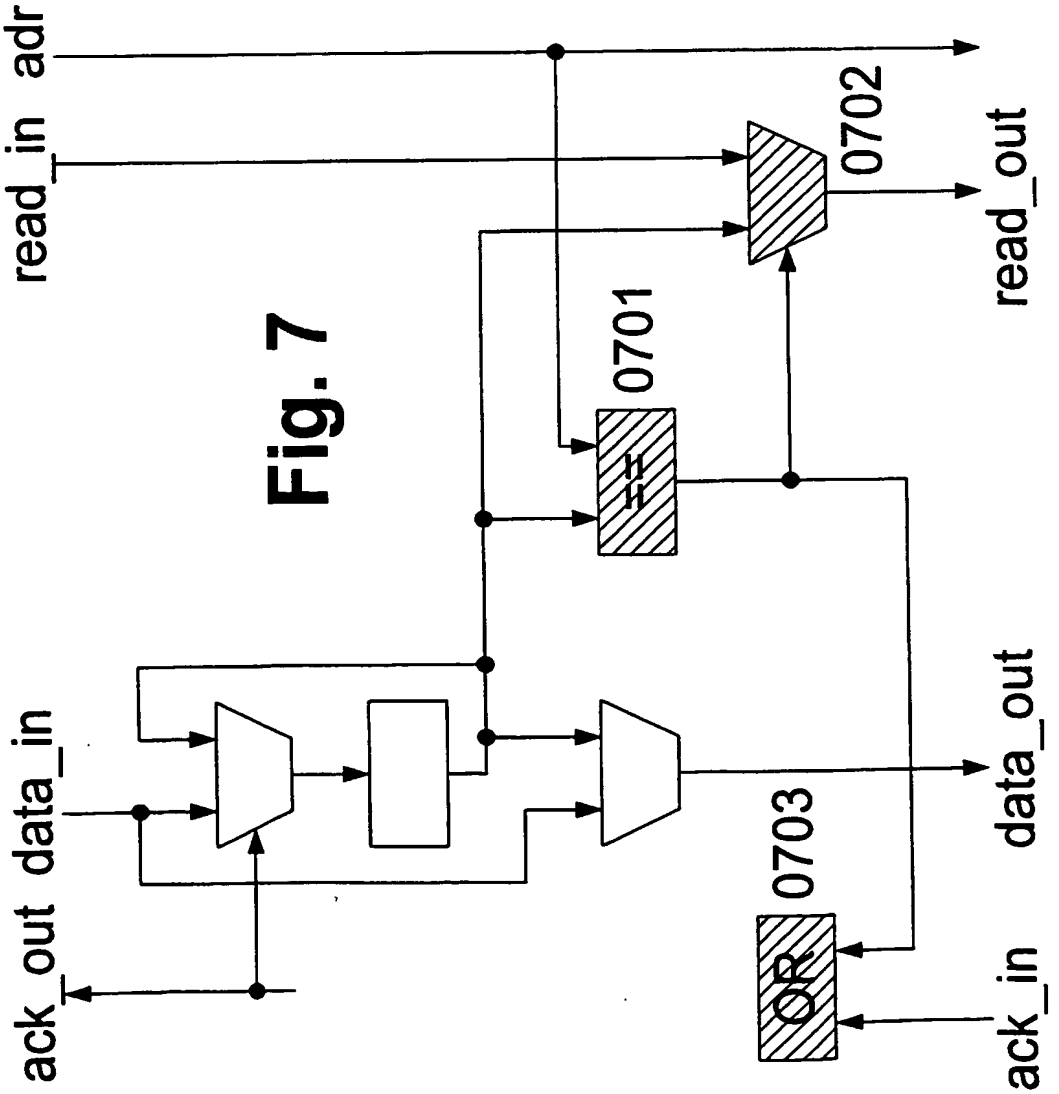

read_in
adr
0702
read_out
0701
ack_out
data_in
0703
OR
data_out
ack_in

Fig. 7

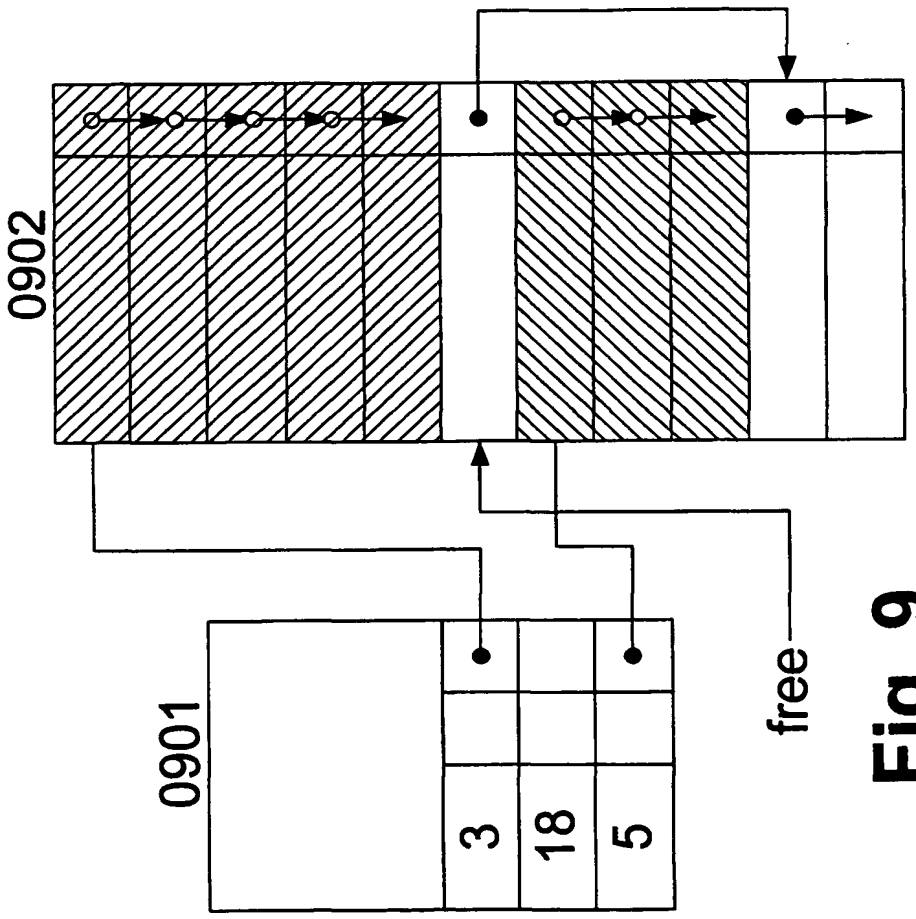


Fig. 9

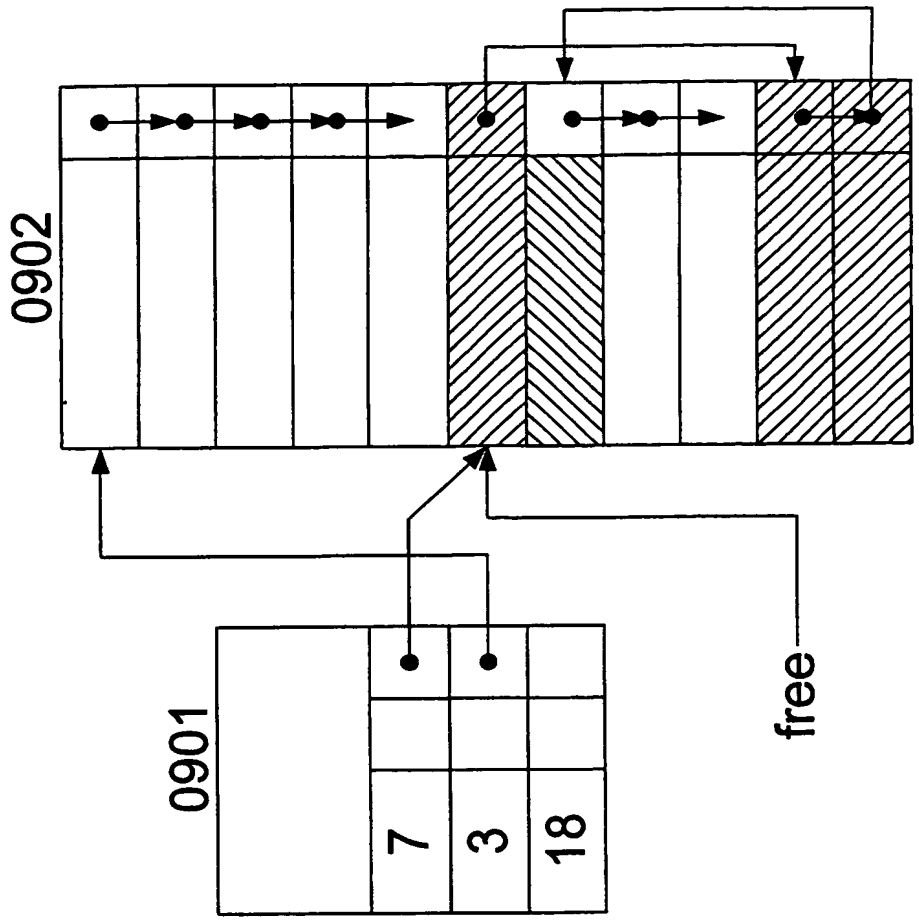


Fig. 10b

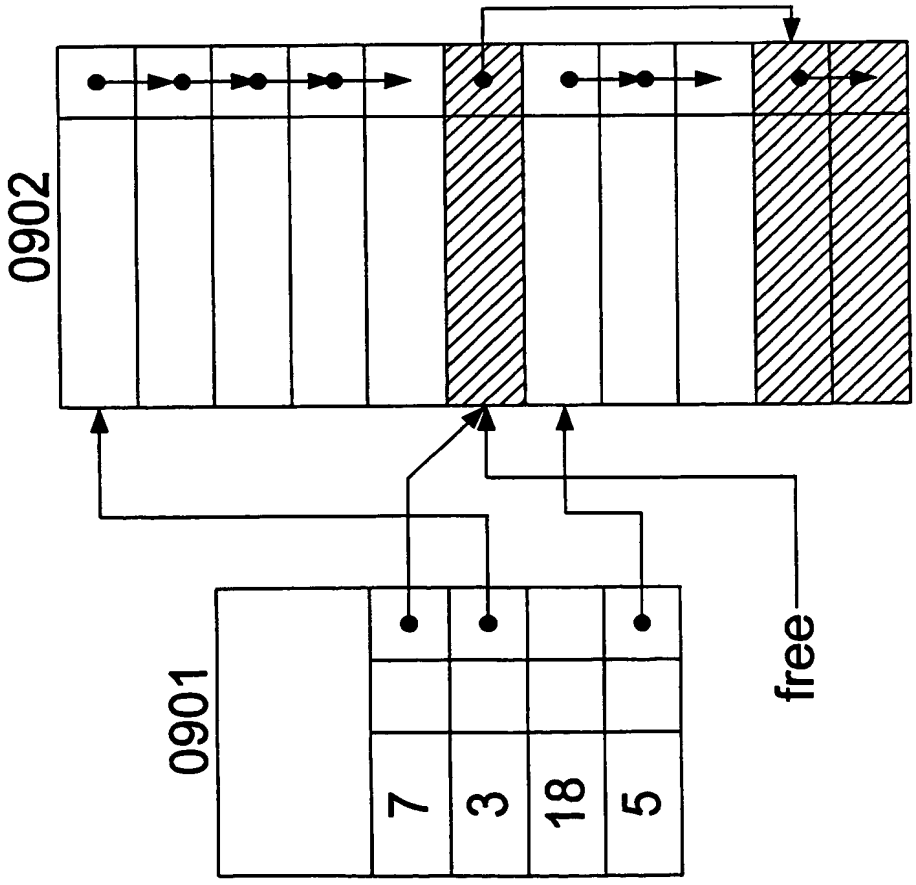


Fig. 10a

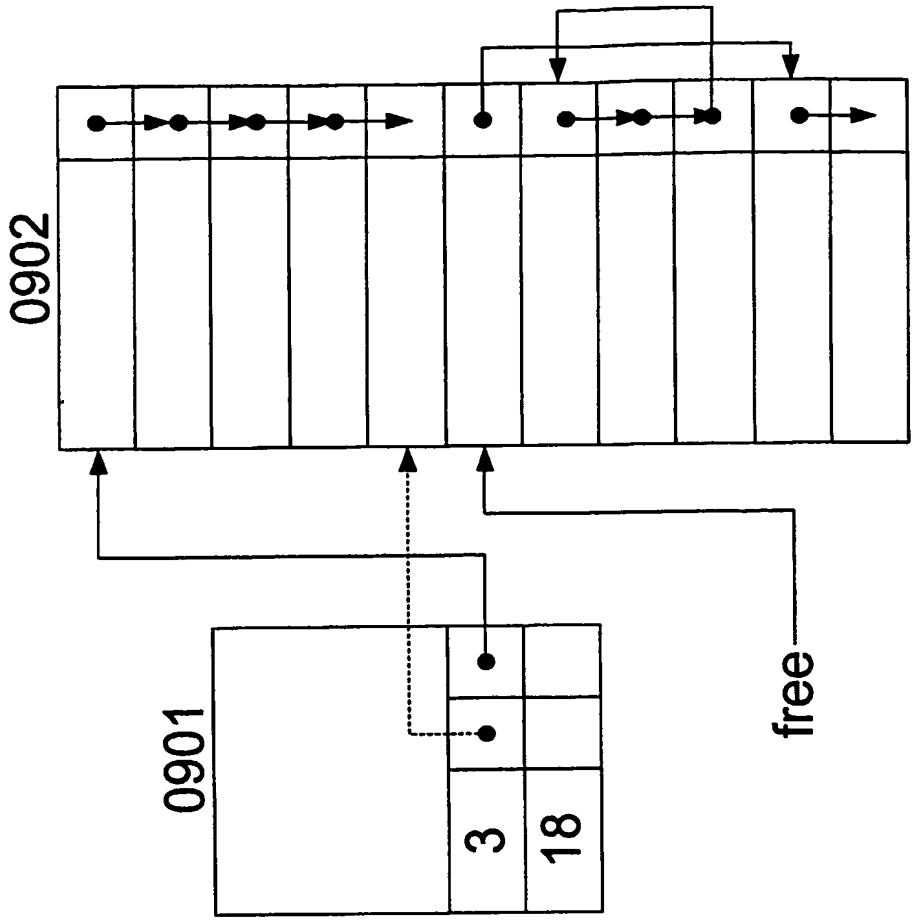


Fig. 11b

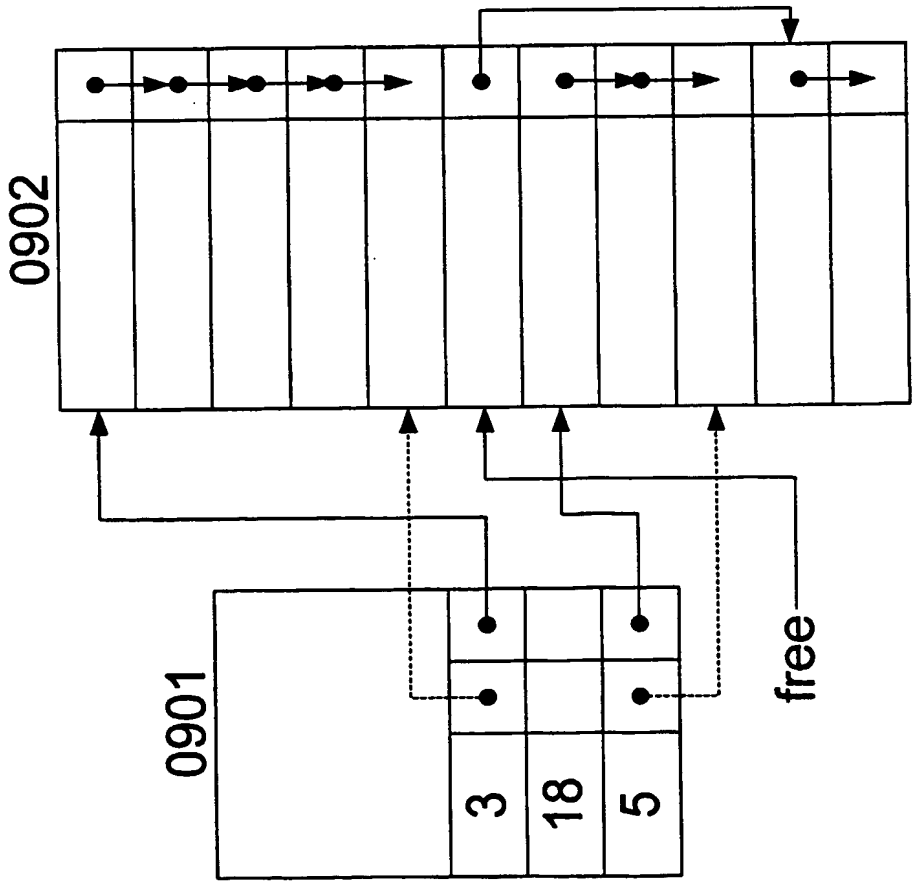


Fig. 11a